-Perform the following hex computations (leave the result in hex):

a) 1234 +9876
b) 0FFF - 0F34
c) 100 - 1
d) 0FFE – 1

==

How many hexadecimal digits in:

a) Byte b) word c) double word

==

- Assuming a 16-bit two's complement format, determine which of the values below are positive and which are negative.

a) 0ABCD b) 1024 c) 0DEAD d) 0ADD e) 0BEEF
f) 8 g) 05AAF h) 0FFFF i) 0ACDB j) 0CDBA
k) 0FEBA l) 35 m) 0BA n) 0ABA o) 0BAD
p) 0DAB q) 4321 r) 334 s) 45 t) 0E65 u) 0BEAD
v) 0ABE w) 0DEAF x) 0DAD y) 9876

==

- What three components make up Von Neumann Machines?

==

7- How many different I/O locations can you address on the 80x86 chip? How many are typically available on a PC?

==

-Convert the following logical addresses to physical addresses. Assume all values are hexadecimal and real mode operation on the 80x86:

a) 1000:1000 b) 1234:5678 c) 0:1000 d) 100:9000 e) FF00:1000
f) 800:8000 g) 8000:800 h) 234:9843 i) 1111:FFFF j) FFFF:10

==

The CPU can access operands (data) in various ways, called addressing modes. In 80x86 there are 7addressing modes , list these modes with an example of each one?

==

- List all of the 80x86 eight bit registers.
- List all the 80x86 general purpose 16 bit registers.
-List all the 80x86 segment registers (those available on all processors).

- Describe the "special purposes" of each of the general purpose registers.

==

- What values appear in the 8086 flags register?

===

In what segment (8086) would you normally place your variables?

===

Consider CS = 2000h ; DS = 1500h ; DI = 0100h ; BX = 0130h ;
SS = 5000h ; SP = 0250h ; BP = 1400h ; AX = 4C69h ;
DX = 8855h ; CX= 6632h

(i S memory .

(ii) Find the logical and physical address of a data stored at offset address BX .

(iii) Find the logical and physical address of a data stored at offset address BP.

(iv) Find the contents of destination and the physical address accessed by each of the following instructions:

- MOV [Bp+100], AX
- MOV SS: 5[BX][DI], DX
- MOV [0200h], CL

What are The addressing mode have used in each of the above instruction.

(v) What do the change in contents of the stack segment and stack segment register after the execution of the following instructions:

PUSH BX
PUSH DX
POP AX

No change will occurs at stack segment register (SS reg)

XX	XX	XX	XX

==

* remember

Segment register	**CS**	**DS**	**ES**	**SS**
Offset register(s)	**IP**	**SI, DI, BX**	**SI, DI, BX**	**SP, BP**

Segment Override:

MOV AL,[BX] DS:BX however MOV AL,ES:[BX]

MOV AX,[BP] SS:BP however MOV AX,DS:[BP]

==

-Provide an example that shows that it requires n+1 bits to hold the sum of two n-bit binary values.

ADC and SBB can be forced to behave exactly like ADD and SUB by inserting some other instruction before ADC and SBB. What instruction must be inserted in front of ADC to make it behave like ADD? In front of SBB to make it behave like SUB?

==

- Provide four different ways to add two to the value in the BX register. No way should require more than two instructions?

==

- Explain the difference between the carry flag and the overflow flag?

==

- What is the difference between a “MOV reg, immediate” instruction and a “LEA reg , address” instruction?

==

- Assume the following register contents: AX = 01FA BX = FF03

What is the NEW value of AX if the instruction " imul bl" is executed?

==

- Assume the following register contents: AX = 001A BX = FFFC

What is the new value of AX if the instruction "idiv bl" is executed?

Remember that AL gets the quotient, and that AH gets the remainder.

==

- Assume the following register contents:

DS: 09A4, SS: 09A1, BX= 0005, BP:001B, EAX = AA2387E4

show how memory is modified for the following instruction mov [bp+4], ax

address	content															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
09A0:0000																
09A0:0010																
09A0:0020																
09A0:0030																
09A0:0040																

==

- For each of the following sums, indicate if unsigned overflow (carry flag), signed overflow (two’s complement) occur (neither, one or the other, or both)

a. F0h + FFh

NEITHER **Only Unsigned overflow** Only Signed Overflow Both

F0 + FF = EFh, sets carry flag. As signed: Negative + Negative = Negative, no signed overflow.

b. 90h + A0h

NEITHER Only Unsigned overflow Only Signed Overflow **Both**

90h + A0h = 30h, sets carry flag, As signed: negative + negative = positive, so signed overflow.

c. 10h + A0h

NEITHER Only Unsigned overflow Only Signed Overflow Both

10h + A0h = B0h, carry flag not set, as signed: postitive + negative cannot overflow.

d. 50h + 40h

NEITHER Only Unsigned overflow **Only Signed Overflow** Both

50h + 40h = 90h. carry flag not set, as signed: positive + positive = negative, so signed overflow.

e. 20h + 30h

NEITHER Only Unsigned overflow Only Signed Overflow Both

20h + 30h = 50h. Carry flag not set, as signed: positive + positive = positive, so no signed overflow.

===

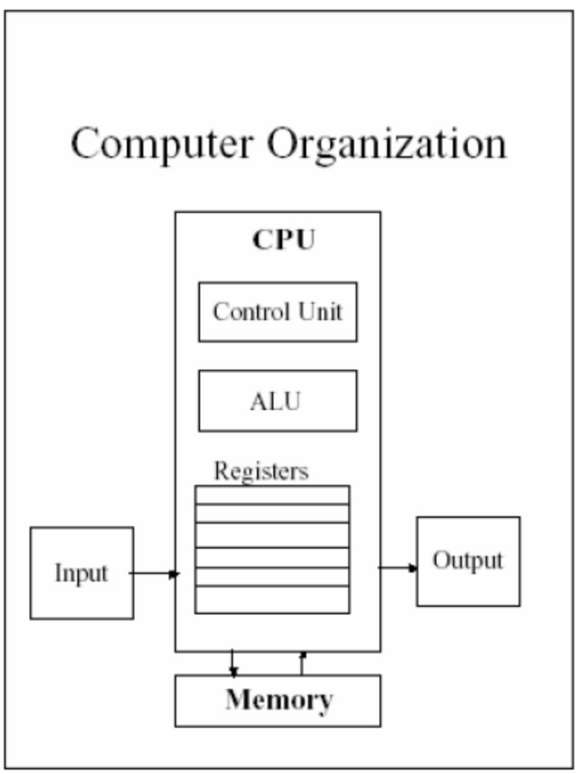
- The instruction "add ax, bx" will do a 16 bit add of AX = AX + BX. How could I do a 16 bit add if I only have 8 bit registers? Write a two instruction sequence that will do a 16 add of AX = AX+BX but you can ONLY use registers AH, AL, BH, BL in your instructions.

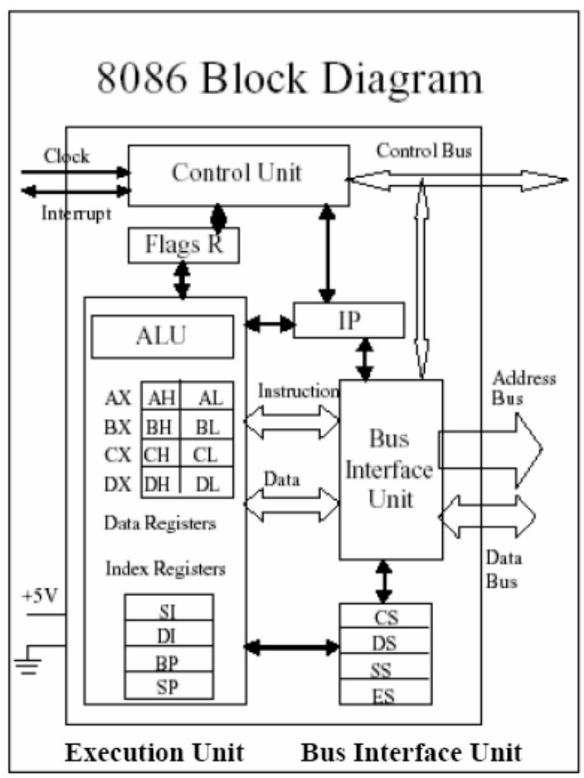
===

- Explain the difference between the following two instructions:

a. mov bx, 0200h b. mov bx, [0200h]

- draw a block diagram of computer organization and 8086 INTERNAL ORGANIZATION





===

- which of the following instruction is correct which is not :

```
MOV BX,14AFH ;    move 14AFH into BX
MOV SI,2345H ;    move 2345H into SI
MOV DI,2233H ;    move 2233H into DI
MOV CS,2A3FH ;    move 2A3FH into CS
```

```
MOV DS,CS ;          move the content of CS into DS
MOV FR,BX ;          move the content of BX into FR
MOV DS,14AFH ;       move 14AFH into DS
MOV AL,123H ;
MOV AX,3AFF21H ;
```

==

Remember :

Values cannot be loaded directly into (CS,DS,SS and ES)

```
MOV AX,1234H ;       load 1234H into AX
MOV SS,AX ;          load the value in AX into SS
```

==

- using the inforamtion below, show the changes of the memory content (stack segment) after excute each instruction of the following: H FIRST

SP=1236, AX=24B6, DI=85C2, and DX=5F93,

```
PUSH AX
PUSH DI
PUSH DX
```

SS:1230

SS:1231

SS:1232

SS:1233

SS;1234

SS;1235

SS:1236

==

- assume that the stack is shown below, and SP=18FA, show the contents of the stack and registers as each of the following instructions is executed.

```
POP CX
POP DX
POP BX
```

SS:18FA
SS:18FB
SS:18FC
SS:18FD
SS;18FE
SS;18FF
SS:1900

===

- Show how the flag register is affected by the addition of 38H and 2FH.

MOV BH,38H ;BH=38H
ADD BH,2FH ;BH = BH + 2F = 38 + 2F= 67H

38	0011 1000
+ 2F	0010 1111
67	0110 0111

===

- Show how the flag register is affected by the following addition

MOV AX,34F5H ;AX =34F5H
ADD AX,95EBH ;AX = CAE0H

34F5	0011 0100 1111 0101
+ 95EB	1001 0101 1110 1011
CAE0	1100 1010 1110 0000

===

```
; This program adds 5 unsigned byte numbers.
.MODEL SMALL
.STACK 64
.DATA
COUNT       EQU   05
DATA        DB    125,235,197,91,48
            ORG   0008H
SUM         DW    ?
.CODE
MAIN:       MOV AX, @DATA
            MOV   DS,AX
            MOV   CX,COUNT          ;CX is the loop counter
            MOV   SI,OFFSET DATA    ;SI is the data pointer
            MOV   AX,00             ;AX will hold the sum
BACK:       ADD   AL,[SI]           ;add the next byte to AL
            ADC   AH,00             ;add 1 to AH if CF =1
            INC   SI                ;increment data pointer
            DEC   CX                ;decrement loop counter
            JNZ   BACK              ;if not finished, go add next byte
            MOV   SUM,AX            ;store sum
            MOV   AH,4CH
            INT   21H               ;go back to DOS
            END   MAIN
```

==

```
; This program is an example for Multiword addition
.MODEL SMALL
.STACK 64
.DATA
DATA1       DQ     548FB9963CE7H
            ORG    0010H
DATA2       DQ     3FCD4FA23B8DH
            ORG    00020H
DATA3       DQ     (?)
.CODE
MAIN:       MOV AX, @DATA
            MOV    DS,AX
            CLC                          ;clear carry before the first addition
            MOV    SI,OFFSET DATA1       ;SI is the data pointer for operand1
            MOV    DI,OFFSET DATA2       ;DI is the data pointer for operand2
            MOV    BX,OFFSET DATA3       ;BX is the data pointer for the sum
            MOV    CX,04                 ;CX is the loop counter
BACK:       MOV    AX,[SI]       ;move the first operand to AX
            ADC    AX,[DI]       ;add the second operand to AX
            MOV [BX],AX                  ;store the sum
            INC    SI                    ;point to next word of operand1
            INC    SI
            INC    DI                    ;point to next word of operand2
            INC    DI
            INC    BX                    ;point to next word of sum
            INC    BX
            LOOP   BACK                  ;if not finished, continue adding
            MOV    AH,4CH
            INT    21H                   ;go back to DOS
            END    MAIN
```

-Analyze the following program:

```
DATA_A DD 62562FAH
DATA_B DD 412963BH
RESULT DD ?
….. …..
MOV AX,WORD PTR DATA_A                  ;AX=62FA
SUB AX,WORD PTR DATA_B                  ;AX=AX – 963B
MOV WORD PTR RESULT,AX                  ;save the result
MOV AX,WORD PTR DATA_A +2               ;AX=0625
SBB AX,WORD PTR DATA_B +2               ;SUB 0412 with borrow
MOV WORD PTR RESULT +2,AX               ;save the result
```

Note: PTR (Pointer) Directive is used to specify the size of the operand. Among the options for size are BYTE, WORD, DWORD and QWORD.

==

- Assume the following memory contents at the start of each of the following instructions

65 E2

Address	Contents															
09A0:0000	C5	67	A5	00	12	BC	34	BB	E4	72	09	A3	29	01	D4	CE
09A0:0010	FE	89	02	D8	A4	8A	7C	DD	90	3C	9B	83	65	19	F6	8A
09A0:0020	A7	CC	9A	BD	8E	90	2C	59	1C	90	0E	13	8C	39	58	C6
09A0:0030	76	D7	CA	FF	D8	71	18	24	40	A8	2C	76	93	C5	0F	9E
09A0:0040	82	A6	54	2E	9A	20	0A	98	E4	A0	0E	25	38	29	2C	86

Assume the following register contents at the START of each of the following instructions.
ES: 09A0, DS: 09A0, SS: 09A1
AX = E265, DX = 73A2, CX = 0000, SP = 0018, BP=002E
Give the value of the affected register OR affected memory location after each instruction.
IF MEMORY IS MODIFIED, you MUST show the modified locations on the ABOVE memory map! LIST ALL registers that are affected by the instructions except for the flag registers.

a. Push ax
b. sar dl , 2
c. shr dl,2
d. shl al,1
e. xor dl, al
f. or dl, F0h
g. not dl
h. pop dx
i. mul dl
j. imul dl
k. and dl, 22

==

- Assume the same register contents/memory contents as above. For EACH of the following two instruction sequences, tell if the jump is TAKEN or NOT TAKEN.

a. cmp al,dl
   jne there
b. cmp al,dl
   jl there
c. cmp al,dl
   ja there
d. cmp ax,dx
   jg there
e. cmp ax,dx
   jb there
f. test al,1
   jnz there
g. add al,dl
   jnc there
h. add al,dl
   js there
i. add al, 40h
   jno there

==

- Register AL has an 8 bit value ($b_7b_6b_5b_4b_3b_2b_1b_0$). Use a single logical instruction to change the contents of AL to ($b_700000b_1b_0$). Bits B7, B1, B0 unchanged, other bits set to zero.

==

- Register AL has an 8 bit value ($b_7b_6b_5b_4b_3b_2b_1b_0$). Use a single logical instruction to change the contents of AL to ($1b_6b_5b_4b_311b_0$). Bits B7, B2,B1, set to '1', other bits unchanged.

===

- Write a subroutine that will return the number of bytes in a string. The string is terminated by a '0' byte (count DOES NOT INCLUDE the '0' byte), and the starting address of the string is passed to the subroutine via the DS:SI register. The count should be returned as zero if the string is 'empty' (first byte is zero). The count should be returned in the AL register (maximum number of characters will be 255).

```
Strlen      proc

Strlen      endp
```

===

- Write a subroutine that will return the maximum 16-bit SIGNED integer from an array of integers. On subroutine entry, register SI will point to the start of the array (each element is 16 bits), and register CX will have the number of integers in the array. The maximum value should be returned in the AX register. An example call to this procedure is shown below:

```
.data
      Myarray     dw    -45, 1000, -34, 1500, 20, 60
      Count    equ 6                               ; six elements in the array
.code
  mov ax,@data
  mov ds,ax

          mov cx, count
          lea si, [myarray]
          call findmax
          …..
          …..

          Findmax      proc
                                               ;;; will assume that array always has at least 1 element

                   mov ax, 8000h               ;; get most negative 16-bit value into ax
             lp1:  cmp ax, [si]
                   jge  skip
                   mov ax, [si]                ;; ax is less than [SI], get memory value
           skip:   lea si, [si+2]              ;; increment pointer by 2 bytes
                   loop lp1                    ;; cx has count
                   ret
      findmax       endp
```

===